University of Pittsburgh

Algorithms and Data Structures 2
CS 1501

Pitt SCI 5 YEARS

Spring 2023

Sherif Khattab

ksm73@pitt.edu

(Slides are adapted from Dr. Ramirez's and Dr. Farnan's CS1501 slides.)

Announcements

- Upcoming Deadlines
  - Lab 6: Tuesday 3/14 @ 11:59 pm
  - Homework 8: this Friday @ 11:59 pm
  - Assignment 2: this Friday @ 11:59 pm
    - Support video and slides on Canvas
- Talk by candidate faculty
  - This Wednesday 3/15 @ 10 am at 5317 Sennott Square
  - Donuts will be served!

Previous lecture

- LZW Compression and expansion

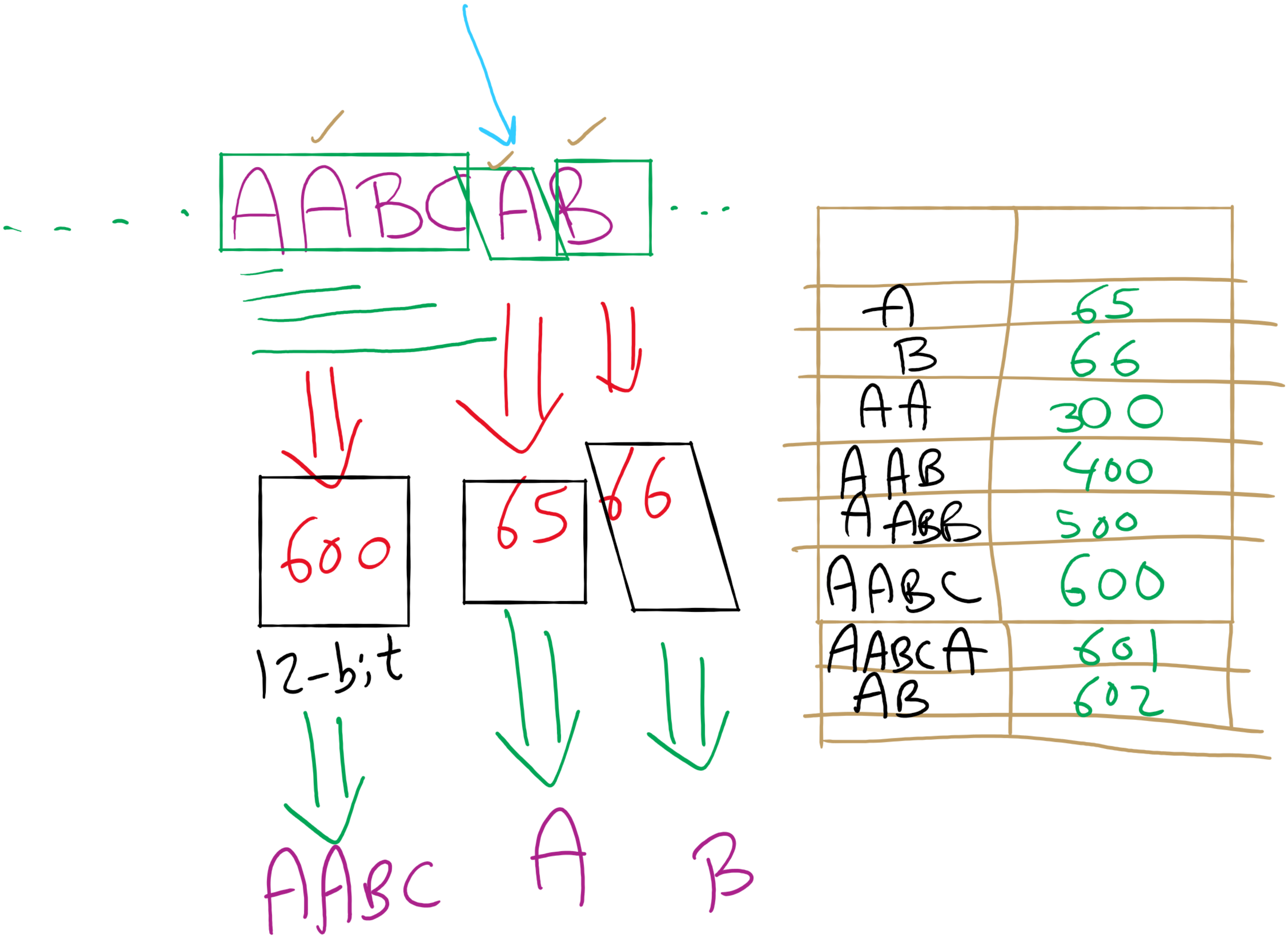
This Lecture

- LZW example and corner case
- Shannon's Entropy
- LZW vs. Huffman
- Burrows-Wheeler Compression Algorithm

LZW Compression

- Both compression and expansion construct the same codebook!
  - Compression stores character string → codeword
  - Expansion stores codeword → character string
  - They contain the same pairs in the same order
    - Hence, the codebook doesn’t need to be stored with the compressed file, saving space

LZW Example


AABCAB
600
65
66
12-bit
AABC
A
B
A 65
B 66
AA 300
AAB 400
AABB 500
AABC 600
AABCA 601
AB 602

Just one tiny little issue to sort out...

- Expansion can sometimes be a step ahead of compression...
  - If, during compression, the (pattern, codeword) that was **just added** to the dictionary is **immediately used** in the next step, the decompression algorithm will not yet know the codeword.
  - This is easily detected and dealt with, however

LZW corner case example

- Compress, using 12 bit codewords: AAAAAA

Cur	Output	Add
A	65	AA:256
AA	256	AAA:257
AAA	257	--

- Expansion:

Cur	Output	Add
65	A	--
256	AA	256:AA
257	AAA	257:AAA

LZW Corner Case

AABBB

65 65 66 258

A A B BB

A	65
AA	256
AB	257
BB	258

65	A
66	B
256	AA
257	AB
258	BB

$2^{|Codeword|}$

Prev output + 1^{st} character of Prev output

LZW implementation concerns: codebook

- How to represent/store during:
  - Compression
  - Expansion
- Considerations:
  - What operations are needed?
  - How many of these operations are going to be performed?
- Discuss

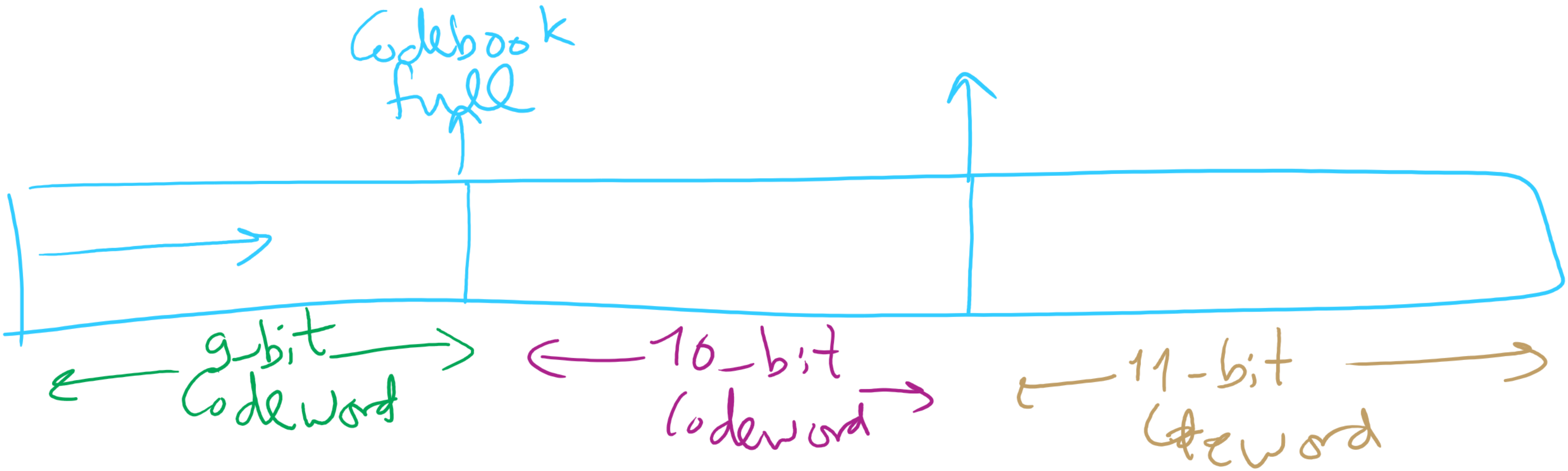
Further implementation issues: codeword size

- How long should codewords be?
  - Use fewer bits:
    - Gives better compression earlier on
    - But, leaves fewer codewords available, which will hamper compression later on
  - Use more bits:
    - Delays actual compression until longer patterns are found due to large codeword size
    - More codewords available means that greater compression gains can be made later on in the process

Variable width codewords

- This sounds eerily like variable length codewords…
  - Exactly what we set out to avoid!
- Here, we're talking about a different technique
- Example:
  - Start out using 9 bit codewords
  - When codeword 512 is inserted into the codebook, switch to outputting/grabbing 10 bit codewords
  - When codeword 1024 is inserted into the codebook, switch to outputting/grabbing 11 bit codewords…
  - Etc.

Adaptive Codeword Size



Even further implementation issues: codebook size

- What happens when we run out of codewords?
  - Only 2^n possible codewords for n bit codes
  - Even using variable width codewords, they can't grow arbitrarily large…
- Two primary options:
  - Stop adding new keywords, use the codebook as it stands
    - Maintains long already established patterns
    - But if the file changes, it will not be compressed as effectively
  - Throw out the codebook and start over from single characters
    - Allows new patterns to be compressed
    - Until new patterns are built up, though, compression will be minimal

Can we reason about how much a file can be compressed?

- Yes! Using Shannon Entropy

Information theory in a single slide...

- Founded by Claude Shannon in his paper “A Mathematical Theory of Communication”
- **Shannon Information** is a measure of the **unpredictability of information content**
  - Example: which is more unpredicatble?
    - a character that occurs with probabillity 0.5 or
    - a character that occurs with probability 0.25
    - which should have more entropy?

Shannon Infomation

- Shannon Information of a message m
  - $I(m) = -1 * \log_2 Pr(m)$ bits
  - $Pr(m)$ is the probability of message m
- Examples:
  - $Pr(c1) = 0.5 \rightarrow I(c1) = -1 * \log_2(0.5) = -1*-1 = 1$ bit
  - $Pr(c2) = 0.25 \rightarrow I(c2) = -1*\log_2(0.25) = -1*-2 = 2$ bits
  - $Pr(c3) = 1/2^{100} \rightarrow I(c3) = -1*\log_2(2^{-100}) = -1*-100 = 100$ bits

Shannon's Information Entropy

- ***Shannon's Entropy*** is a key measure in information theory
  - Slightly different from thermodynamic entropy
- Entropy of an information source (e.g., a file)
  - $H = \text{sum}_{\text{all unique messages } m}\ Pr(m) * I(m)$
  - average of Shannon's information of all unique messages
- **Entropy per bit** = H / file size in bits

Entropy of a file

How can we determine the probability of each character in the file?

- if it depends only on file contents
  - *Pr(c)* = *f(c)* / file size
- However, may also depend on receiver and sender contexts and their world knowledge

Implications on Lossless Compression

- By losslessly compressing data, we represent the same information in less space
  - entropy of original file = entropy of compressed file
- On average, a lossless compression scheme cannot compress a message to have more than 1 bit of entropy per bit of compressed message

Entropy applied to language

- **Entropy of a language:** the average number of bits required to store a letter of the language
- Uncompressed, English has between 0.6 and 1.3 bits of entropy per letter
- Entropy of a language * length of message in characters = amount of information contained in that message

The showdown you've all been waiting for...

HUFFMAN vs LZW

- In general, LZW will give better compression
  - Also better for compressing archived directories of files
    - Why?
      - Very long patterns can be built up, leading to better compression
      - Different files don't "hurt" each other as they did in Huffman
        - Remember our thoughts on using static tries?

So lossless compression apps use LZW?

- Well, gifs can use it
  - And pdfs
- Most dedicated compression applications use other algorithms:
  - DEFLATE (combination of LZ77 and Huffman)
    - Used by PKZIP and gzip
  - Burrows-Wheeler transforms
    - Used by bzip2
  - LZMA
    - Used by 7-zip
  - brotli
    - Introduced by Google in Sept. 2015
    - Based around a " ... combination of a modern variant of the LZ77 algorithm, Huffman coding[,] and 2nd order context modeling ... "

Is there a univeral compression algorithm?

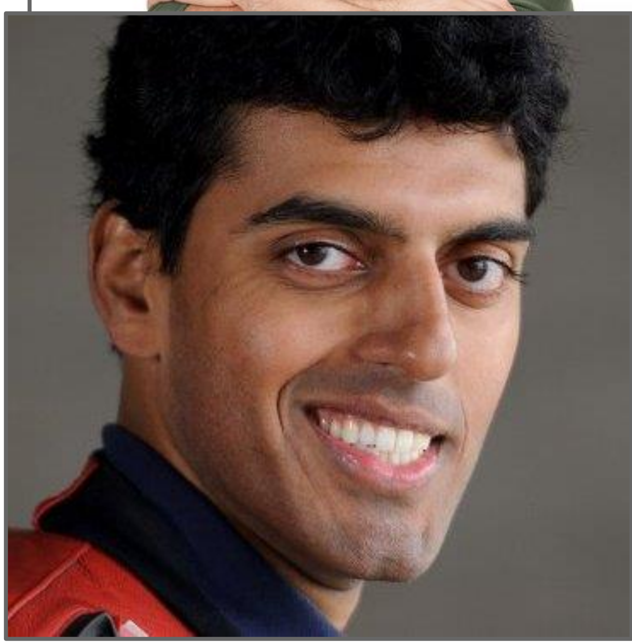
- That can compress every file and any file?
- Nope!
- No algorithm can compress every bitstream
  - Assume we have such an algorithm
  - We can use to compress its own output!
  - And we could keep compressing its output until our compressed file is 0 bits!
    - Clearly this can't work
- Proofs in Proposition S of Section 5.5 of the text

What about the best compression algorithm for a given file?

- Nope!
- This problem is undecidable
- Example:
  - A Fibonacci sequence of one billion numbers can be compressed by a program to generate Fibonacci numbers

A final note on compression evaluation

- "Weissman scores" are a made-up metric for Silicon Valley (TV)



Burrows-Wheeler Data Compression Algorithm

- **Best** compression algorithm (in terms of compression ratio) **for text**
- The basis for UNIX's **bzip2** tool

Adapted from: https://www.cs.princeton.edu/courses/archive/spr03/cos226/assignments/burrows.html

BWT: Compression Algorithm

- Three steps
  - Burrows-Wheeler Transform
    - Cluster same letters as close to each other as possible
  - Move-To-Front Encoding
    - Convert output of previous step into an integer file with **large frequency** differences
  - Huffman Compression
    - Compress the file of integers using Huffman Compression

BWT: Expansion Algorithm

- Apply the inverse of compression steps in reverse order
  - Huffman decoding
  - Move-To-Front decoding
  - Inverse Burrows-Wheeler Transform